What is claimed is:

1. A method comprising:

relocating a compiled code block associated with a software application;

wherein said relocating is performed responsive to hardware event information gathered during current execution of the software application; and

wherein said relocating is performed during current execution of the software application.

2. The method of claim 1, further comprising:

selecting the compiled code block responsive to occurrence of a trigger condition during current execution of the software application.

3. The method of claim 2, wherein:

the hardware event information indicates that the trigger condition has occurred during current execution of the software program.

4. The method of claim 2, wherein:

the trigger condition is a threshold number of hardware performance events.

5. The method of claim 4, wherein:

the trigger condition is a threshold number of instruction miss events.

6. The method of claim 2, wherein:

the trigger condition is a set of hardware criteria.

7. The method of claim 2, wherein:

the trigger condition is a set of hardware and software criteria.

8. The method of claim 1, further comprising:

selecting the compiled code block based on the compiled code block's resource utilization during current execution of the software application.

9. The method of claim 8, wherein:

the hardware event information reflects the compiled code block's resource utilization during current execution of the software program.

10. The method of claim 9, wherein:

the hardware event information further reflects the number of executed method calls performed by the compiled code block.

11. The method of claim 9, wherein:

the hardware event information further reflects the number of times the compiled code block has been called during execution of the software program.

12. The method of claim 9, wherein:

the hardware event information further reflects the number of execution cycles consumed during execution of the compiled code block.

13. The method of claim 1, further comprising:

relocating a virtual method table associated with the software application during current execution of the software application.

14. The method of claim 13, wherein:

said hardware event information includes data miss information.

15. The method of claim 1, wherein:

said relocating is performed on as as-needed basis independent of garbage collection.

16. The method of claim 1, wherein said relocating further comprises:

moving the compiled code block to a new location within a code region; and

patching address references in the code region to reflect the new location of the compiled code block.

17. The method of claim 16, wherein:

patching address references further comprises patching address references such that invocation of the relocated compiled code does not generate a trap.

18. The method of claim 16, further comprising:

patching a call stack to reflect the new location.

19. The method of claim 16, further comprising:

patching a virtual method table to reflect the new location.

20. A method, comprising:

dynamically relocating a program element; and

invoking a just-in-time compiler to patch address references associated with the relocated program element.

21. The method of claim 20, wherein:

the program element is a compiled code block.

22. The method of claim 20, wherein:

the program element is a virtual method table.

23. The method of claim 20, further comprising:

generating hardware event information during current execution of a software program with which the program element is associated.

24. The method of claim 23, wherein dynamically relocating a program element further comprises:

determining whether to relocate the program element based on the hardware event information.

25. The method of claim 24, wherein determining whether to relocate the program element further comprises:

determining whether the hardware event information indicates that a trigger condition has been met.

26. The method of claim 24, wherein:

the program element is a compiled code block; and

determining whether to relocate the program element further comprises determining whether calls to the compiled code block have generated at least a predetermined number of instruction miss events.

27. The method of claim 24, wherein:

the program element is a virtual method table; and

determining whether to relocate the program element further comprises determining whether accesses to the virtual method table have generated at least a predetermined number of data miss events.

28. The method of claim 23, wherein dynamically relocating a program element further comprises:

determining a new location for the relocated program element based on the hardware event information.

29. The method of claim 23, wherein:

the hardware event information includes branch history information.

30. The method of claim 23, further comprising:

generating profile information based on the hardware event information.

31. The method of claim 30, wherein:

the profile information includes a call graph.

32. An article comprising:

a machine-readable storage medium having a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

dynamically relocating a program element; and

invoking a just-in-time compiler to patch address references associated with the relocated program element.

33. The article of claim 32, wherein the storage medium has instructions, which if executed by a machine, cause the machine to further perform operations comprising:

generating hardware event information during current execution of a software program with which the program element is associated.

34. The article of claim 33, wherein the instructions that cause the machine to dynamically relocate a program element further include instructions, which if executed by a machine, cause the machine to further perform operations comprising:

determining whether to relocate the program element based on the hardware event information.

35. The article of claim 33, wherein the instructions that cause the machine to dynamically relocate a program element further include instructions, which if executed by a machine, cause the machine to further perform operations comprising:

determining a new location for the relocated program element based on the hardware event information.

36. The article of claim 34, wherein the instructions that cause the machine to determine whether to relocate the program element further include instructions, which if executed by a machine, cause the machine to perform operations comprising:

determining, based on the hardware event information, whether a trigger condition has been met during current execution of the software program.

37. The article of claim 36, wherein the instructions that cause the machine to determine whether a trigger condition has been met further include instructions, which if executed by a machine, cause the machine to perform operations comprising:

determining whether calls to the program element have generated at least a predetermined number of instruction miss events.

38. The article of claim 36, wherein the instructions that cause the machine to determine whether a trigger condition has been met further include instructions, which if executed by a machine, cause the machine to perform operations comprising:

determining whether accesses to the program element have generated at least a predetermined number of data miss events.

39. The article of claim 36, wherein the instructions that cause the machine to determine whether a trigger condition has been met further include instructions, which if executed by a machine, cause the machine to perform operations comprising:

determining whether execution of the program element has utilized at least a predetermined quantity of execution resources.

40. An apparatus, comprising:

a compiled code region to store compiled native codes; and

a code manager to dynamically modify layout of the compiled code region based on hardware event feedback.

41. The apparatus of claim 40, wherein:

the hardware event feedback is generated during current execution of a software program.

42. The apparatus of claim 40, wherein:

the code manager is further to determine, based on the feedback, whether the layout of the compiled code region should be modified.

43. The apparatus of claim 40, wherein:

the code manager is further to determine, based on the feedback, a new location for a compiled code block.

44. The apparatus of claim 40, further comprising:

code manipulation logic to patch address references in the compiled code region.

45. The apparatus of claim 40, further comprising:

a virtual method table region.

46. The apparatus of claim 45, wherein:

the code manager is further to dynamically modify layout of the virtual method table region based on hardware event feedback.

47. The apparatus of claim 46, wherein:

the code manager is further to determine, based on the feedback, whether the layout of the virtual method table region should be modified.

48. The apparatus of claim 46, wherein:

the code manager is further to determine, based on the feedback, a new location for a virtual method table.

49. A system comprising:

a processor; and

a memory, wherein the memory further comprises:

a compiled code region;

a code manager to dynamically manage layout of the compiled code region; and

code manipulation logic to patch address references in the compiled code region.

50. The system of claim 49, further comprising:

a virtual method table region.

51. The system of claim 50, wherein:

the code manager is further to dynamically manage layout of the virtual method table region.

52. The system of claim 49, wherein:

the memory is a DRAM.

53. The system of claim 49, wherein:

the code manager is further to dynamically manage layout of the compiled code region based on dynamic hardware event information.

54. The system of claim 53, wherein:

the code manager is further to dynamically manage layout of the compiled code region based on dynamic hardware instruction miss information.

55. The system of claim 51, wherein:

the code manager is further to dynamically manage layout of the virtual method table region based on dynamic hardware event information.

56. The system of claim 55, wherein:

the code manager is further to dynamically manage layout of the virtual method table region based on dynamic hardware data miss information.